The KAPTUR Environment: An Operations Concept

Contract Number NAS5-30680
Task 12

Prepared For:

NASA/Goddard Space Flight Center
Data Systems Technology Division
Greenbelt, MD 20771

August 18, 1989

CTA INCORPORATED
6116 Executive Boulevard, Suite 800
Rockville, MD 20853
(301)816-1200

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

This report presents a high-level specification and operations concept for *KAPTUR*—a development environment based on Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationales. KAPTUR is intended to do what its name implies: to capture knowledge that is required or generated during the development process, but that is often lost because it is *contextual*, i.e., it does not appear directly in the end-products of development. Such knowledge includes issues that were raised during development, alternatives that were considered, and the reasons for choosing one alternative over others. Contextual information is usually only maintained as a memory in a developer's mind. As time passes, the memories become more vague and individuals become unavailable, and eventually the knowledge is lost. KAPTUR seeks to mitigate this process of attrition by recording and organizing contextual knowledge as it is generated.

KAPTUR also seeks to facilitate the application of knowledge to future developments. From the relations between past and ongoing work, developers can fortify their understanding of the current problem and its possible solutions. To support the application of prior knowledge to new problems, KAPTUR must be able to place any new knowledge that it acquires in the context of knowledge it already has. The following table lists some of the types of knowledge that KAPTUR will manage.

- Specifications
- Tutorial explanations of components
- Distinctive features of a system
- Justifications for distinctive features
- Critiques

KAPTUR's knowledge base will contain contextual information normally left implicit.

Each knowledge type in KAPTUR can be related to other knowledge types in various ways. As the knowledge types change—for example, a new modeling technique is introduced—the relations between knowledge types must also evolve. The table below lists some of the relationships that KAPTUR will eventually support. Many of these relationships are dependent on the information's content as opposed to its type. For example, if the same item occurs in different artifacts and receives a different explanation for each occurrence, these explanations should be linked.

- Links between items in specifications (including across description types and specifications)
- Links between items and their explanations, justifications, and critiques
- Links between explanations, justifications, and critiques
- Links between terms in specifications, explanations, justifications and critiques

Information in KAPTUR will be linked by its content.

To support these dynamic, content-dependent relationships, KAPTUR will employ a *hypertext* approach to managing its knowledge base. This approach will result in many links between artifacts in KAPTUR. If two artifacts incorporate a common element or group of elements, then they are linked by this item. If two systems address a common issue in different ways, then they are linked by the issue. KAPTUR will even link systems if they simply use the same terminology—even if the usage is slightly different in the two systems. These links define a *hypermedia* network through which the developer may navigate according to the current requirements. This network is similar to a hypertext network, except that the hypermedia network includes graphical as well as textual objects. The primary advantage of this form of organization is that items are linked according to their content and not according to a predefined set of relations. As the content evolves, so do the relationships.

By linking items in KAPTUR's knowledge base according to content, the relations can be structured to reflect the developer's view of the problem and solution domains. This does not, however, guarantee that the relations will be useful for comparing and contrasting potential solutions to a problem. Two developers, for example, may use identical names for items that have different purposes. Comparing artifacts by following the links between such items may be misleading. To mitigate this problem, KAPTUR must ensure that at least some of the links between artifacts will aid in comparison of alternatives. KAPTUR will do this through an analysis of *distinctive features*: those features of an artifact that differ from one or more accepted models (in the POCC domain, for example, such features include new subsystems and additional functions). Links will be established between artifacts based on their distinctive features. For each distinctive feature of a new artifact, KAPTUR will locate artifacts of the same type with the same feature, and artifacts that incorporate alternatives to the feature, and will link both of these groups to the new artifact.

The remainder of this document describes KAPTUR in more detail. Section 1.1 presents a typical scenario of the use of KAPTUR for exploring alternative control center software achitectures. Section 2 describes how KAPTUR will be used to understand existing designs and to create new ones. Section 3 presents a detailed description of these functions, including the screens that will make up the KAPTUR interface, and the use of the options on each screen.

1.1 A Scenario of the Use of KAPTUR

The idea for KAPTUR grew out of a domain analysis of control center software (see the reports listed in Section 1.2). In the process of comparing software architectures for different mission control center application processors, we found ourselves reverse engineering the rationales for various decisions. These decisions concerned, for example, the inclusion or omission of functions, the grouping of functions, and the levelling of subsystems and components.

This process suggested that a reuse environment should not simply present to the developer a set of alternative architectures that have been used for previous systems: the developer would have no sound basis on which to select one architecture over the others, to merge aspects of several, or to define yet another software architecture. It is, instead, necessary to present the rationales and issues involved in choosing among the alternatives.

Figure 1-1 illustrates how KAPTUR would be used to explore alternative software architectures for a control center application processor. In the center of the diagram there is a knowledge base containing information about the application domain. This includes recommended architecture(s) and information about previously developed systems. In this scenario, the developer has available a set of software requirements, and wants to begin defining an applications processor to meet these requirements. The developer sits down at the KAPTUR workstation and issues a command whose meaning is something like the following:

> I want to develop a control center applications processor. Show me what they look like.

In response, KAPTUR displays the recommended generic architecture (upper right-hand box). In this scenario, there is only one recommended generic architecture because that is the conclusion (to date) of our domain analysis. That conclusion may change as our analysis continues, or as the domain evolves; and certainly at lower levels of the software there will be alternative recommended approaches for different requirements.

Upon examining the recommended architecture, the developer has the following options:

- ACCEPT the recommended architecture
- Examine the DISTINCTIVE FEATURES of the recommended architecture
- Examine ALTERNATIVES to the recommended architecture
- Define a NEW architecture

If the developer selects ALTERNATIVES, KAPTUR displays a list of existing systems that are different from the recommended generic architecture in some significant respect (since the recommended architecture is generic, almost every production system will be different in some respects, if only by instantiating various generic parameters of the recommended architecture). This display is shown in the lower left-hand box of Figure 1-1. The developer can then select one or more of these systems to view their respective architectures. As with the recommended architecture, the developer has the

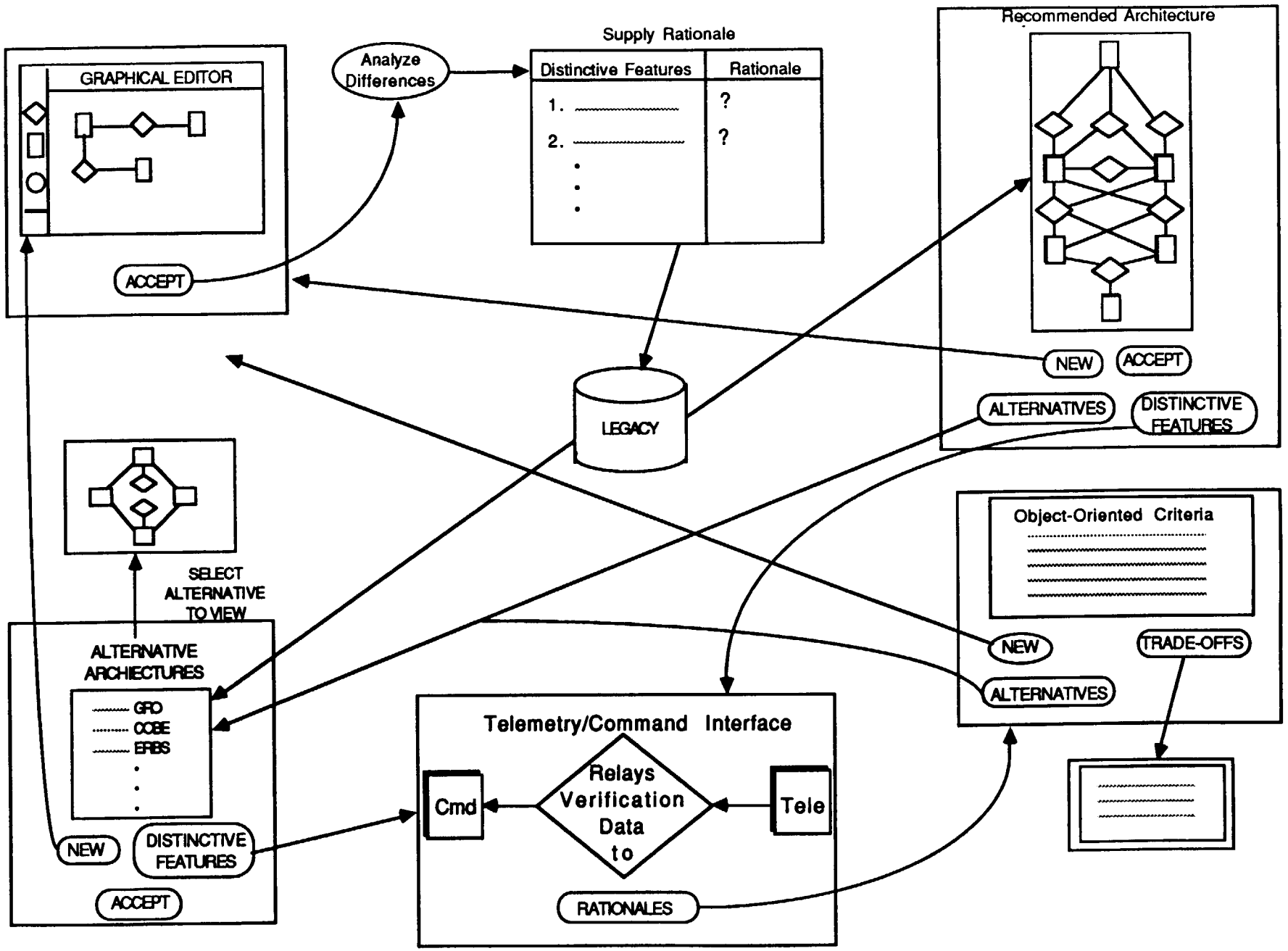

Figure 1-1: KAPTUR may be Used to Explore Alternative Control Center Architectures

option here of ACCEPTing one of the other architectures, or of deciding to define a NEW architecture.

The DISTINCTIVE FEATURES of an architecture are those that are different from common practice or the recommended approach, or that represent a non-trivial decision about a significant issue. It is the knowledge and analysis underlying these decisions that KAPTUR is intended to preserve. Distinctive features may correspond to specific portions of an architecture (e.g., in Figure 1-1, the interface between the Telemetery and Command Subsystems), or they may represent some aspect of the architecture as a whole (e.g., the distribution of initialization functions to all subsystems).

If the developer selects DISTINCTIVE FEATURES, KAPTUR will list the distinctive features of the architecture being displayed, and will allow the developer to select one or more of these features. KAPTUR will then display a representation of the distinctive feature(s). In effect, the developer has the capability to *zoom* into a view of a particular feature of the architecture. This is illustrated in the bottom-middle box in Figure 1-1.

The developer can then examine the RATIONALES for this feature, i.e., the reasoning underlying the decision that the feature represents. In the lower right-hand box in Figure 1-1, we illustrate the rationales as a list of object-oriented design criteria that might underlie the decision. From this screen, the developer can request even more detailed exaplanations, by asking to view the TRADE-OFFS that were considered in reaching the decision. The developer can also ask to see ALTERNATIVES to this decision, i.e., other systems that do not possess this feature because a different decision was made.

If the developer selects NEW (from either the Recommended Architecture or the Alternative Architectures screen), a graphical editor will be invoked to allow the interactive definition of the new architecture. However, the definition of the new archtitecture need not proceed completely from scratch. A clipboard capability in KAPTUR will allow the developer to select portions of the recommended and/or alternative architectures for inclusion in the new architecture.

Once a new architecture has been defined, KAPTUR will perform an automated analysis to determine its distinctive features, i.e., the ways in which it is significantly different from the recommended architecture. For each distinctive feature, KAPTUR will prompt the user to enter one or more rationales justifying the feature. This is shown in the top-middle box (above LEGACY) in Figure 1-1.

The new architecture, together with its rationales, then becomes part of the LEGACY of this domain, and will appear in the ALTERNATIVES list when KAPTUR is next used. This is how the evolution of domain requirements and solutions is captured in the knowledge base. At some point, the differences between the recommended architecture(s) and new systems may become so numerous that a reevaluation of the recommended architectures is necessary. The practical need for knowledge base management therefore drives the ongoing domain analysis process, ensuring that domain models are kept current with present requirements.

1.2 References

The following CTA reports summarize work leading up to this Operations Concept:

Semi-Automatic Development of Payload Operations Control Center Software. Prepared for Computer Sciences Corporation under Contract NAS5-31500, Task 28-11600. October 31, 1988. This document contains additional references to previous work.

Generic POCC Architectures. Prepared for Computer Sciences Corporation under Contract NAS5-31500, Task 28-11600. April 5, 1989.

Generic POCC Architecture: Revised Recommended Refinements and Object-Oriented Interfaces. Prepared for NASA/Goddard SPace Flight Center Data Systems Technology Division under Contract NAS5-30680, Task 12. June 30, 1989.

2 AN OVERVIEW OF KAPTUR

KAPTUR has two primary goals. The first is to help users understand existing software specifications and the reasons they are defined as they are. The second is to help a developer apply existing knowledge in creating new specifications. We are not able to address these goals in their most general forms all at once. Instead, the initial capability will be restricted to the types of specifications that arose in our domain analysis of control center software (see the table, below). That analysis concentrated on the top two or three levels of software architecture specification. The specification models are quite general, however, and can be applied in other contexts as well. Moreover, the operational concept of KAPTUR applies to the full range of artifacts that are produced in the course of software development—only the initial capability is restricted to this set of four types of specifications.

The following sections describe the goals in more detail and present an overview of how KAPTUR will address them.

Understanding Existing Systems. Understanding an existing system involves browsing its associated specifications (requirements, design, code, test plans, etc.) and examining their key features. A developer may browse a specification for one of two reasons. First, the developer may be looking for an explanation of this type of system or this type of specification. Alternatively, the developer may wish to become familiar with a particular system by looking at its specifications. The primary difference between these two forms of browsing is the level of understanding the developer brings to the process. In the first case, the developer lacks background in the domain to which the system belongs, or in the specification technique used. In the second case, the developer is familiar with the domain and the specification technique, but wants to know the details of a specific case, i.e., how a given specification is different from others of the same type (and why). KAPTUR will support both types of developers, and will allow a developer to move freely between these two modes of browsing.

Systems are typically described from various "views." One of KAPTUR's key features will be to allow developers to move arbitrarily between various views, and between levels within a view. Initially, KAPTUR will support a limited number of view types, which are listed in the table below. KAPTUR's features, however, are view-independent. This will allow the set of supported view types to grow to support new system models, new design methodologies, and other application domains. Ultimately, the modeling techiniques behind the various views will be unified in KAPTUR's underlying knowledge base structure.

- Entity Relationship Diagrams (ERDs)
- Data Flow Diagrams (DFDs)
- Entity-Interface Diagrams (EIDs)
- Composition Graphs (CGs)

Specification Types Supported by KAPTUR

The Entity Relationship Diagrams (ERDs) form the basis of a software architecture specification. The ERDs identify the major entities of the system and the ways in which they are related to each other. A software achitecture definition may contain a hierarchy of ERDs, each level representing a more detailed definition of the system. The DFDs show the data flow between entities in the system. When defined to a sufficient level of detail, DFDs may also show the functions performed by each entity. The Entity Interface Diagrams (EIDs) provide a detailed description of dependencies between entities. They define the functions performed by each entity and the functions of other entities upon which each entity depends. The Composition Graphs (CGs) show the end-to-end processing of system-level stimulae.

One reason for a developer to browse the KAPTUR knowledge base is to understand, in general terms, the structure and function of a system or component in a given domain. We want to present the developer with one or more *recommended generic models*. There may be several recommended models, each satisfying different constraints. KAPTUR must be able to support the existence of several recommended solutions, and it must help the developer discern which solution is most applicable to a given situation. To support developer tutorials, each item in a recommended model (or solution) will have a textual description associated with it. The description will explain what the item is, its role in the model, and locations where additional information on the item may be found. KAPTUR will support explanations of other models as well, but only the recommended models are guaranteed to have them.

Feature Browsing. A developer browsing a specification may want to know what distinguishes it from other specifications of the same type. Many distinctive features will follow from the problem the specified system is intended to solve. At the developer's request, KAPTUR will retrieve and display an specification's distinctive features. The features retrieved at any given time will depend on the view that the developer is currently browsing. For example, the features shown at the top-most level of the ERD hierarchy will be different from those shown for a low level DFD. Not every element of a specification corresponds to a distinctive feature. The distinctive features are those that differ from the recommended model or from common practice.

For each distinctive feature of a specification, there will be a corresponding set of rationales. These rationales will explain why the feature is present. The rationales may consist of several levels of justification for the feature. A general justification will present the design principles that the feature promotes and/or violates. A more detailed justification will map this general principle onto the specific elements of the specification whose configuration or definition is in question. The developer may select the level of justification to be viewed, and may move between levels as desired.

Creating New Specifications. The second major use of KAPTUR is to create new specifications. There are three ways in which a developer may do this; these are listed in the table below.

- Specify from scratch
- Build the specification from portions of other specifications
- Use another specification as a template

Techniques for Creating Specifications

To support the specification of software architectures, for example, KAPTUR will provide full architecture editing capabilities. Developers may add new items to existing architectures, as well as delete and modify existing items. These capabilities will also allow the developer to define a new architecture from scratch.

We rarely, however, specify anything really from scratch. A better way is to base the specification on successful accomplishments from the past, i.e., by using existing specifications as templates which may then be modified. The developer thereby *inherits* the results of much of the previous work. For example, if a developer uses an existing architecture as a template, and modifies only a single entity, then the new architecture inherits all of the other diagrams in the architecture. It also inherits all of the explanations, justifications and features attached to these diagrams. The effort required to develop these items would be saved.

The idea of using template specifications may be extended to include using several specifications as templates. For example, portions of several architectures may be selected and then combined to form a new architecture. KAPTUR will support this approach by means of a ***clipboard*** system. Developers may add to the clipboard as they browse the existing specifications, selecting those portions they desire. They may then invoke the specification editor using either another specification as a template or starting from scratch, and paste items from the clipboard into the specification. Items may be pasted at any level and in any diagram type within the specification. When an item is placed on the clipboard or pasted from it, the item will inherit its previous links. The major drawback of this method is that it may be difficult to achieve consistency between the collected items as they are added to the new specification. Naming conventions and interfaces will probably have to be adjusted.

Capturing Design Understanding. As parts of a new system are specified, KAPTUR will automatically establish links between them and other specifications (or portions of them). These links will enable future developers to understand how the new specification relates to its predecessors.

In addition to capturing relations between specifications, KAPTUR also has facilities for capturing deeper levels of understanding: explanations, justifications, and critiques of distinctive features. Such information may be associated with any item in a specification, but it *must* be associated with the distinctive features. For each such feature it is important that the developer provide some justification; otherwise, future developers will be unable to understand why this feature was adopted and under what conditions it should be adopted again.

Consistency Checking. Consistency between the views of a system must be maintained. Therefore, if a view is modified, then all related views may need to be modified to keep them consistent with the modified view. Ultimately, KAPTUR will assist the developer in producing consistent specifications by marking inconsistencies as they are introduced. KAPTUR will inform the developer of these inconsistencies at the end of an editing session. It will then be the developer's responsibility to make the necessary changes.

Configuration Control. Using existing specifications as templates allows developers to inherit work from past developments, but it also entails some risks. Developers should be careful when building from models that are not generally accepted, or that are incomplete or unjustified. KAPTUR will perform a certain amount of configuration management in order to reduce the potential for such problems. Each specification in

KAPTUR's knowledge base will be assigned a status attribute with four possible values, which are listed in the table below.

- Incomplete: author intends to further edit the specification
- Unjustified: author has completed the specification but has not provided justifications for distinctive features
- Complete: author has justified the distinctive features
- Accepted: the specification has been approved for future reuse

Levels of Model Validity in KAPTUR

The first level (i.e., incomplete) indicates that the specification is still being defined. The second level indicates that the definition is completed but that its unique features have not been justified. The third level indicates that the distinctive features have been justified, but that the community of potential reusers has not yet accepted the specification as a viable model for reuse. The final level indicates that the specification is accepted as a viable reusable model. When a developer tries to base a new system on an unaccepted model, KAPTUR will issue a warning. The developer may still use the unaccepted specification as a template, but should be on guard for possible problems.

3 THE KAPTUR INTERFACE

In this section we describe the user interface for KAPTUR's first prototype. Section 3.1 presents some of the general issues associated with the interface, and Section 3.2 presents the windows that make up the interface.

3.1 General Interface Issues

KAPTUR's first prototype will be hosted on a Sun Workstation and will be based on the X Windows system and NASA's TAE Plus tool. Many of the terms used in this section are from this domain. Some of these are *mouse, select, cursor, window,* and *field.* We assume readers are already familiar with these terms. If not, they should consult a document that discusses them, such as *An Introduction to TAE Plus.*

In keeping with the TAE Plus tradition the interface presented here is for the most part *modeless.* This means that the options available in a given window are not dependent on selections made in other windows. There are only two exceptions to this. The first one is *dialog boxes.* Dialog boxes are windows that inform users of mistakes they have made and that request users to provide missing parameters. For example, one dialog box might advise a user to save part of an edit session before moving on to another edit session. Another dialog box might request the user to specify which item is to be edited before an editor is invoked.

The second exception to modeless operation has to do with the difference between editing and viewing an item. After opening a window to view an item, a user may then choose to jump to other items in the system. When jumping from a View Window (see below), the user may not modify any subsequently visited items. If, on the other hand, the jump starts from an Edit Window, the user may edit the items subsequently visited.

We have tried to make the interface as friendly as possible. To this end, each window in the environment includes a standard set of options. These options provide guidance on how to use the windows, and allow the user to tailor the environment to his own preferences. Each of these options is described in the table below.

- *Help:* The *help* option provides an explanation of each field and button in the window. If the user selects the entire window (by clicking on the border), KAPTUR will provide a high-level description of the window's purpose. If the user selects a field or button in the window, KAPTUR will provide a detailed explanation of how the item is used.

- *Close:* The *close* option closes the current window. If the user has not invoked any options in the window, the *close* option will return the user to the previous dialog location (i.e., window).

Options Available on Every Window

- *Window manipulation:* The *window* option in each window allows the user to move and resize windows in the environment.

- *Where:* The *where* option invokes the Context Window (see the following section). This window allows the user to determine how the current window was arrived at (i.e., the series of options invoked), and it allows the user to return to a previous point in the session.

Options Available on Every Window (cont.)

KAPTUR will provide a concept of *development directory* in order to support organizational groups that have their own sets of conventions, standard practices, priorities etc. Since the purpose of KAPTUR is to promote the reuse of the development knowledge, and thus encourage uniformity of development approach wherever possible (even between development groups), a means of exporting and importing knowledge from/to development directories will be provided.

KAPTUR will provide the user with a single *Clipboard* for a session. The Clipboard is a location where the user can store miscellaneous pieces of information, obtained from various sources during a session. The user may also retrieve these information items at any time in order to "paste" them into an artifact being created.

3.2 The KAPTUR Interface

This section presents the windows that make up KAPTUR's user interface. The discussion for each window includes a description of the options available. The windows form a hierarchy reflecting the order in which they invoke each other. At the top of the hierarchy is the Home Window, which provides access to the top-level KAPTUR commands.

Home Window. KAPTUR's Home Window is shown in Figure 3-1. It allows the user to specify the current development directory and to invoke KAPTUR's top-level functions. The Home Window appears whenever a user invokes KAPTUR. The user's name (i.e., *user id*) is automatically placed in the *User Name* field upon invocation. It is important for the user to verify this name because the user id, along with the development directory, will determine the user's access rights and privileges.

Whenever the user invokes an option from the Home Window, KAPTUR first checks to see that the information needed to perform the request has been provided. If it has not, the user is prompted accordingly. The most common prompt is for the user to name an artifact (usually a specification) on which the option is to be performed—see the View option, below. Once the needed information has been provided, KAPTUR checks to see that the information is valid and that the user has the necessary access privileges. If any problems are detected, KAPTUR will display a dialog window explaining the problem and then cancel the option. For example, if a user tried to edit a specification for which he did not have over-write permission, KAPTUR would display a dialog window describing the access problem and then terminate the edit option. The user can then select another option.

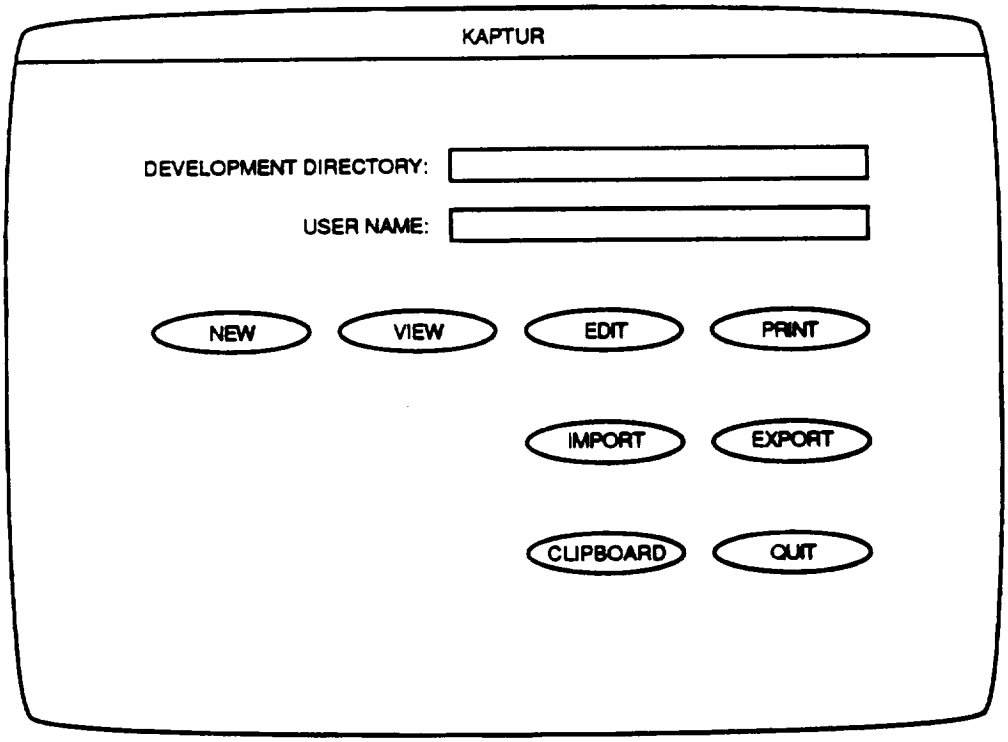

Figure 3-1: Home Window Fields and Functions

The following table describes the options available from KAPTUR's Home Window.

- *Development Directory:* Displays the current development directory. This field may be modified to change the development directory.
- *User Name:* The current user's id, as specified at log in. This field may not be modified.
- *View:* The *view* button invokes the View Window. However, before this window may be invoked, the user must select the specification to be viewed. This is done using the Selection Window, which is displayed prior to the View Window.
- *Edit:* The *edit* button invokes the Edit Window. However, before this window may be invoked, the user must select the specification to be viewed. This is done using the Selection Window, which is displayed prior to the View Window.
- *New:* The *new* button invokes the New Window. From this window the user identifies what specifications are to be used as templates for the new specification.
- *Print:* The *print* button first invokes the Selection Window, in which the user identifies the specification to be printed. The Print Window is then invoked to determine the print options to be used.
- *Import:* The *import* button invokes a dialog window which allows the user to specify a source development directory and a specification name to be copied to the current development directory. To use this option, the user must have the necessary access rights.
- *Export:* The export button first invokes the Selection Window, in which the user identifies the specfication to be exported. A dialog window is then invoked for the user to name a destination project directory. The specification will be copied to this destination. To use this option, the user must have the necessary access rights.

Home Window Fields and Functions

Each invocation of KAPTUR (i.e., the Home Window) establishes a new user session. Any windows created during the session are considered sub-windows of the session and are only visible to other sub-windows of the same session. This means that the user may only move, cut, and paste between windows associated with the same session. To avoid confusion, a user should not engage in simultaneous KAPTUR sessions.

The Home Window will remain on the screen throughout a user's session, and its features may be used at any time in the session. This means that *a user may have several specification view and editing windows open at the same time*. Several windows may even be used to show different aspects of the same specification. The maximum number

of windows will be restricted only by the host system's available memory. This feature makes it easier to compare specifications and transfer information between them.

Selection Window. The Selection Window allows the user to select a specification as an argument to a KAPTUR option. Selections may be made in terms of the specification name and/or identification of items in the specification. The window's layout is presented in Figure 3-2, and its options are described in the table below.

- *Specification List:* This is a scrollable text window that displays the specifications available in the current development directory. An item in the window may be selected by moving the cursor over it and clicking the left mouse button.
- *Select:* When the *select* button is pressed (using the mouse), it causes the name of the currently selected specification to be returned to the calling window. The Select Window is then closed.
- *Find:* The *find* button invokes the Find Window, which allows the user to locate specifications according to the items they contain. When the Find operation is completed, the specification name it returns appears as selected in the Specification List.

Selecting an Architecture

Find Window. Figure 3-3 shows the Find Window. The Find Window is an extension of the Selection Window. It allows the user to find a specification according to the items it contains. For the first prototype the only item types supported are entities, relations, and functions. This list may be expanded as other diagram types are added to the tool.

To specify a search, the user fills in one or more fields in the Find Window. The search is then invoked using the *find* button. If the search is successful, a dialog box containing the list of the located specifications will appear. The user may then select a specification from this list. The last selection made will be displayed in the *last found* field. If the user is happy with this selection, he may invoke the *select* button which will close the Find Window and return the selection to the calling Selection Window; otherwise, he may specify another search and continue the find process.

The table below contains a description of the fields and buttons on the Find Window.

- *Entity:* This is a modifiable text field in which the user may enter an entity name. This name is then used as part of the next search.
- *Function:* This is a modifiable text field in which the user may enter a function name. This name is then used as part of the next search.

Finding a Specification by its Properties

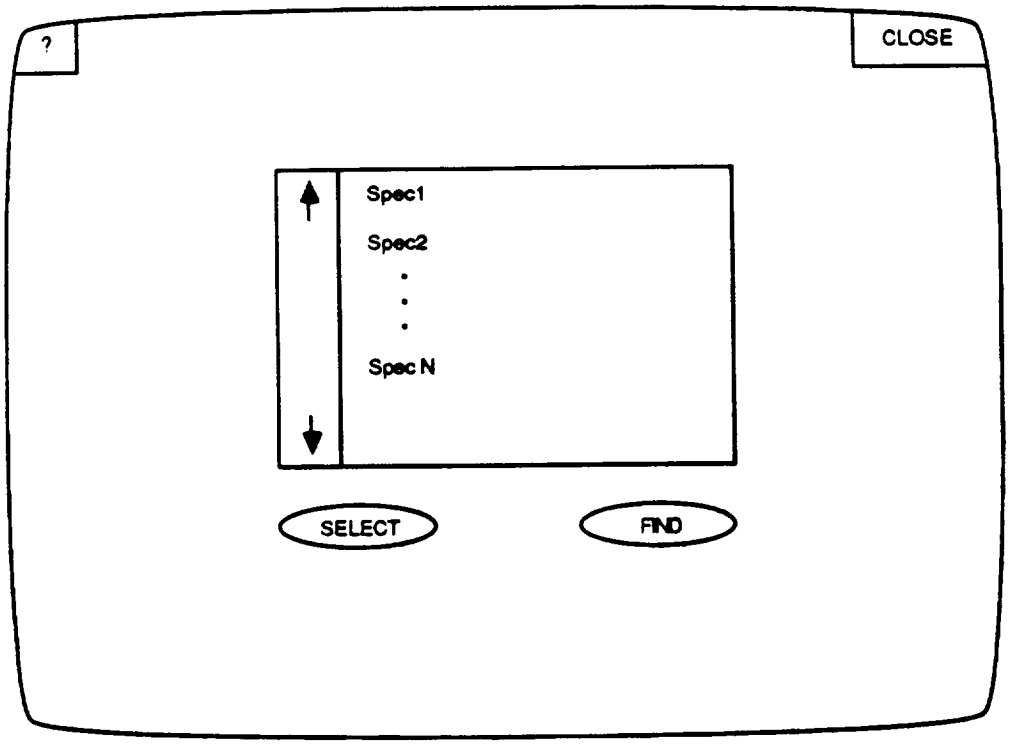

Figure 3-2: Selecting a Specification

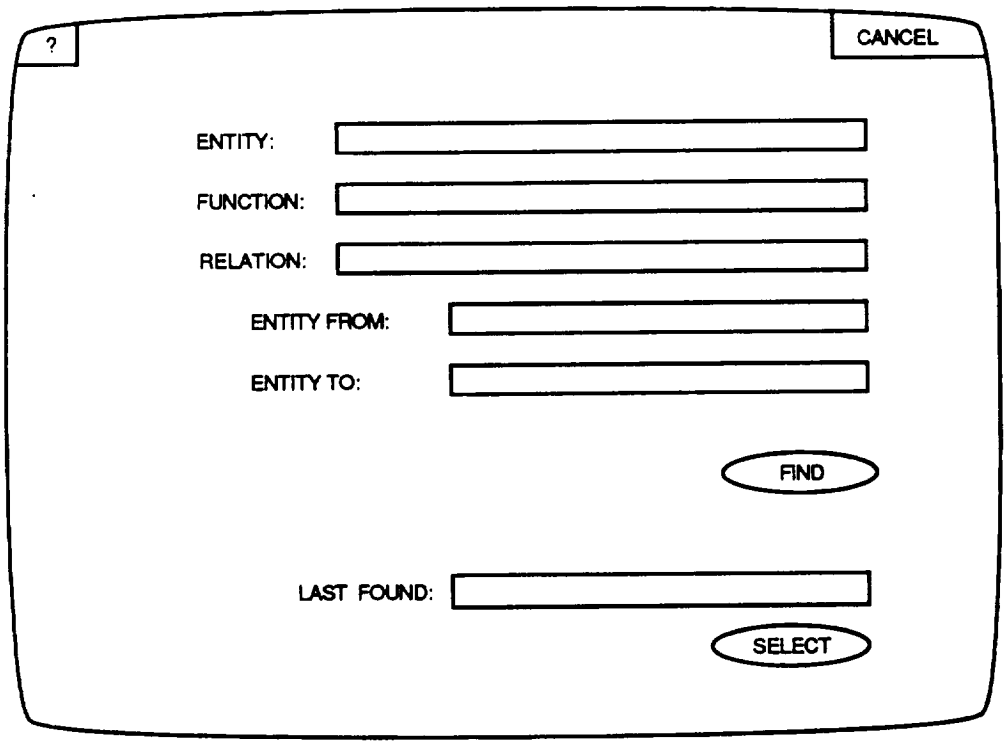

Figure 3-3: Finding a Specification by its Properties

- *Relation:* The *relation* field contains several sub-fields, each of which is a modifiable text field in which the user may enter text. The fields allow the user to specify the name of a relationship, the entity at its head and the entity at its tail. All or one of these fields may be used in locating a specification having the given relationship.

- *Find:* The *find* button invokes the find function, which uses the information in the text fields to search for specifications having the given properties.

- *Last Found:* The *last found* field is a non-modifiable text field that displays the specification last selected by the user. This value is returned to the Selection Window when the *select* button is invoked.

- *Select:* The select button closes the Find Window and returns the current value of the *last found* field to the Selection Window.

Finding a Specification by its Properties (cont.)

View Window. The View Window allows the user to study the details of a specification. At any one time, the window will only display a single view of the specification (i.e., diagram), but the user may invoke the window's options to move to other views. For the first prototype, the only views supported are E-R diagrams with function lists for each entity. Other diagram types will be added in the future.

Figure 3-4 presents an example of a View Window. Its options are described in the table below.

- *Specification View:* The *specification view* displays the current view of the specification. If no view has been specified, then the top-level E-R diagram is displayed. In the first prototype, *specification view* will be a scrollable text screen that displays a textual representation of the diagram. This will be changed to a graphics editor in the future.

- *Up and Down:* The *up* and *down* buttons allow the user to navigate through a specification's diagram hierarchy. *Up* causes the current diagram in the *specification view* to be replaced by its parent diagram. *Down* first invokes a dialog window which prompts the user to select an entity, and then replaces the current diagram in the *specification view* with the selected entity's diagram (if it exists).

- *Functions:* The *functions* button first invokes a dialog window which prompts the user to select an entity. Then it creates a text window and places the selected entity's associated functions in the text window.

- *Features:* The *features* button invokes a text window which displays the distinctive features of the current view. The features window will provide a means of moving from features to their rationales, critiques, and explanations.

Viewing a Specification

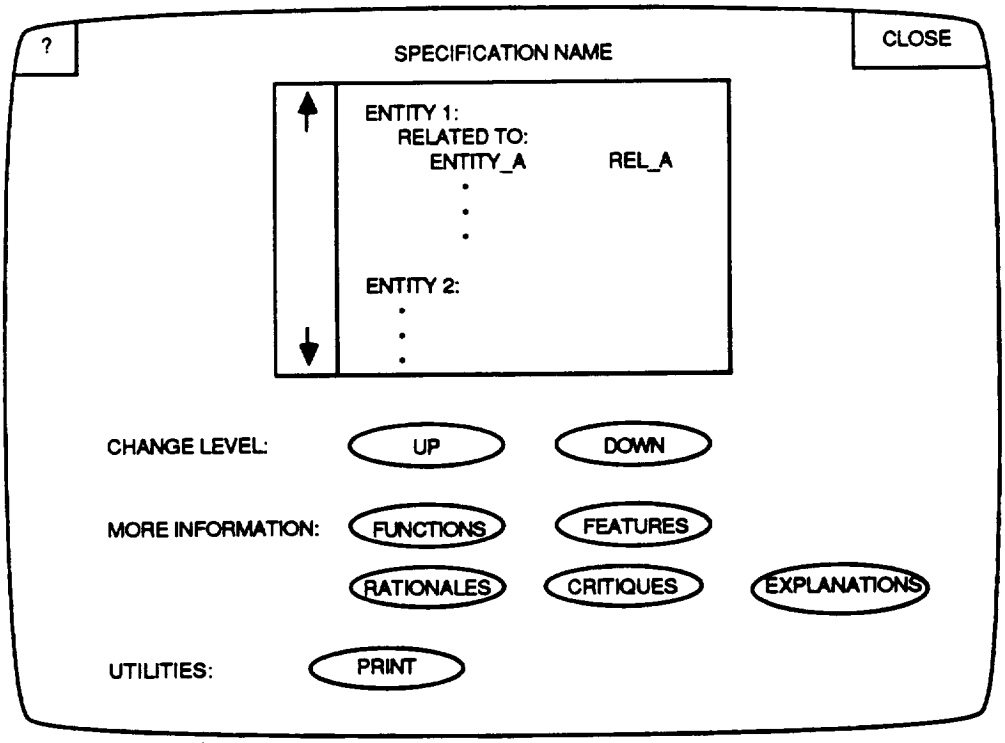

Figure 3-4: Viewing a Specification

- *Rationales:* The *rationales* button invokes a text window which displays the rationales associated with a view. Unlike the rationales associated with a feature, the rationales associated with a view may refer to the view as a whole or to portions of a view.

- *Critiques:* The *critiques* button invokes a text window which displays the critiques associated with a view. Unlike the critiques associated with a feature, the critiques associated with a view may refer to the view as whole or to portions of a view.

- *Explanations:* The *explanations* button first invokes a dialog box requesting the user to select an item (i.e., an entity, relationship, or function) in the current view. If the user selects an item, then a text window containing an explanation of the item is displayed. If no item is selected, a text window explaining the view as a whole is displayed.

- *Print:* This button allows the user to print information from the current view. It invokes the Print Window, which allows the user to specify the options to be used in printing the view.

Viewing a Specification (cont.)

Edit Window. The Edit Window contains options that allow the user to modify the items in a specification. The two most common types of modification are insertion and alteration. To insert items, the user moves the cursor to the place where the insertion is desired and begins typing. To alter items, the user selects a piece of text by clicking the left mouse button at the beginning of the text and the right mouse button at the end of the text and then issues the appropriate alteration commands (e.g., delete). The user may also move items by pasting them on the Clipboard, deleting them from their current location, and then pasting them from the Clipboard to a new location. Copies may be made in a similar fashion. The user may also insert new items from the Clipboard into the specification.

Figure 3-5 shows an example Edit Window, and the table below describes the options available in it. Many of the options are similar to the options in the View Window, except that they allow the user to modify the items instead of just viewing them. One major difference pertains to requests for an item that does not exist in the current view. When this occurs in a View Window, a dialog box explaining the problem is invoked and the option is terminated. In the Edit Window a blank or partially filled template is displayed, so that the user may create the item.

- *Current Specification View:* This is the current view of the specification, including any modifications the user may have made to the view. If no view has been specified, then the top-level ER diagram is displayed. In the first prototype, the *current specification view* will be a scrollable text screen that displays a textual representation of diagrams. This will be changed to a graphics editor in the future.

KAPTUR's Specification Editing Capabilities

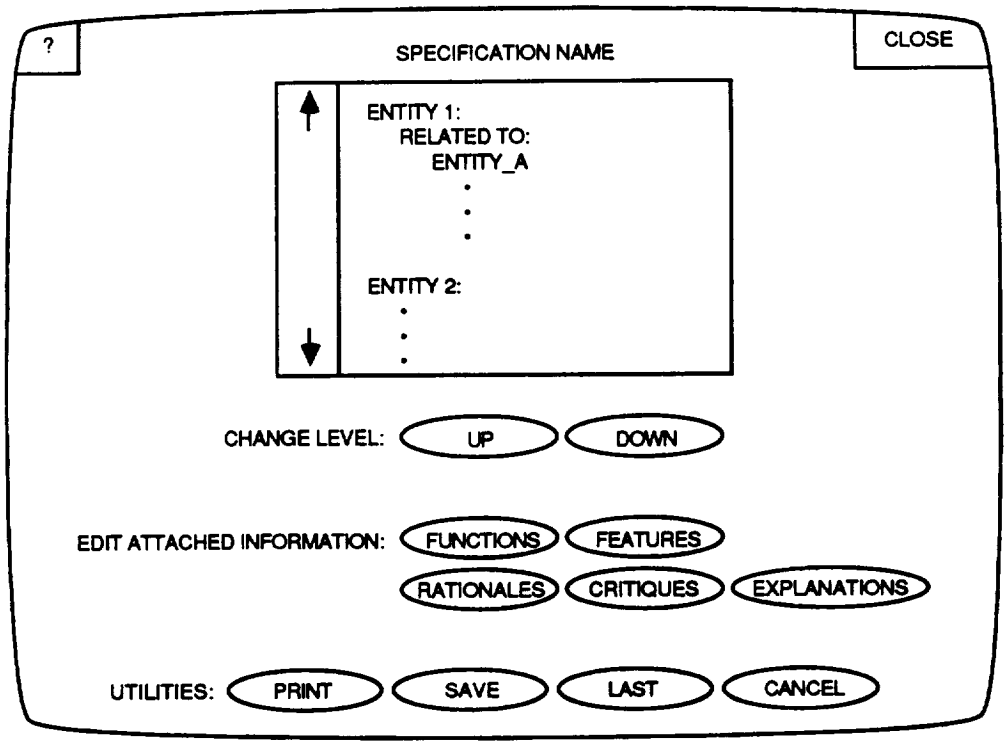

Figure 3-5: KAPTUR's Specification Editing Capabilities

- *Up and Down:* The *up* and *down* buttons allow the user to navigate through a specification's diagram hierarchy. *Up* causes the diagram in the *current specification view* to be replaced by its parent diagram. *Down* first invokes a dialog window which prompts the user to select an entity, and then replaces the diagram in the *current specification view* with the selected entity's subdiagram. If the entity does not have a subdiagram, then a template diagram is created for it and this template is placed in the *current specification view*. Before moving to a new diagram, the up and down options check to make sure the *current specification view* has been saved. If it hasn't, a dialog box is invoked that prompts the user to do so.

- *Functions:* The *functions* button first invokes a dialog window which prompts the user to select an entity, and then it displays a text window listing the functions associated with the entity. Functions listed in the window may be modified as desired.

- *Features:* When the *features* button is selected, an analysis of the current view is performed. This analysis determines the distinctive features of the current view.. Descriptive identifiers of any distinctive features found are displayed in a text window. The user may browse but not modify this list. The user may, however, create and /or modify explanations, critiques and justifications associated with the distinctive features.

- *Rationales:* The *rationales* button invokes a text window which displays the existing rationales associated with the current view. The user may modify these or provide additional rationales.

- *Critiques:* The *critiques* button invokes a text window which displays the existing critiques associated with the current view. The user may modify these or create additional critiques.

- *Explanations:* The *explanations* button first invokes a dialog box requesting the user to select an item (i.e., an entity, relationship, or function) in the current view. If the user selects an item, a text window explaining the selected item is displayed. If no item is selected, the text window contain an explanation of the view as a whole. The user may modify or add to the existing explanations.

- *Print:* The *print* button allows the user to print information from the current view. It invokes the Print Window, in which the user can specify options to be used in printing.

- *Save:* The *save* button saves the current view in the KAPTUR knowledge base.

- *Last:* The *last* button loads the last saved version of the current specification view into the Edit window.

- *Cancel:* The *cancel* button loads the version of the current view that existed before the current edit session began. This in effect ignores any saves that have been performed during the edit session.

KAPTUR's Specification Editing Capabilities (cont.)

New Window. The New Window is activated whenever the user selects the *new* button from the Home Window. It i used to create a new specification. The user provides a name for the new specification, and may identify an existing specification that will serve as a template for the new one. KAPTUR copies the template and invokes an Edit Window with the new copy. The user may then modify the new specification as desired. If no template is specified, an empty Edit Window is created instead.

Figure 3-6 shows a New Window. Its options are described in the table below.

- *Specification Name:* The *specification name* field is a modifiable text field in which the user specifies the new specification's name.
- *Based On:* The *based on* field is a modifiable text field that contains the template specification's name. The user may type a name into the field or use a Selection Window to choose the template. This latter option is performed using the *select template* button (see below).
- *Select Template:* The *select template* button invokes a Selection Window in which the user may select a template for the new specification. See the description of the Selection Window (above) to see how this is done.
- *Define Specification:* The *define* specification button invokes an Edit Window in which the new specification will be defined. If a template name appears in the *based on* field, this template is copied and placed in the Edit Window. If none is specified, the Edit Window will be empty.

Establishing a Template for a New Specification

In addition to using a complete specification as a template, the user may also use portions of a specification. This is done by yanking (i.e., copying) portions from other specifications and placing them on the Clipboard, then moving them from the Clipboard into the Edit Window for the new specification.

Print Window. The Print Window allows the user to specify options for printing information contained in or associated with a specification. Complete specifications are printed from the Home Window. Individual views are printed from a View or Edit Window. Each print option may be turned on or off by clicking on the *yes/no* radio button beside each option. Figure 3-7 shows an example Print Window. An explanation of the print options is presented in the table below.

- *View:* If the *view* button is set to yes, the specification view(s) are printed.
- *Features:* If the *features* button is set to yes, the distinctive features associated with the specification are printed.
- *Rationales:* If the *rationales* button is set to yes, the rationales associated with the specification and its features are printed.

Specification Print Options

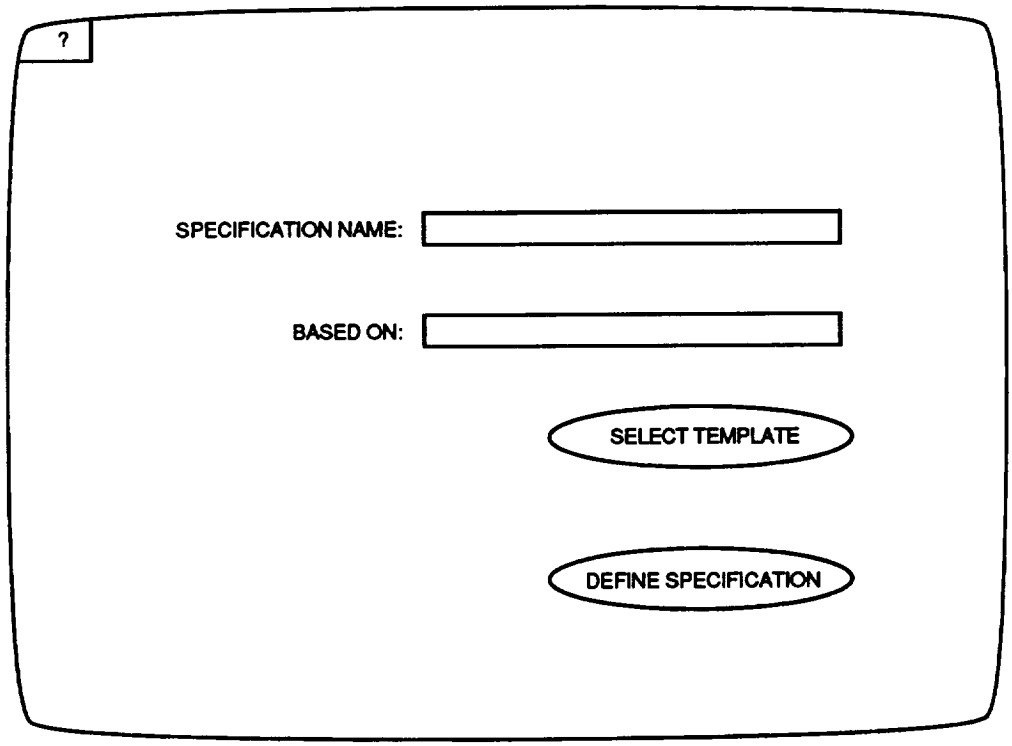

Figure 3-6: Establishing a Template for a New Specification

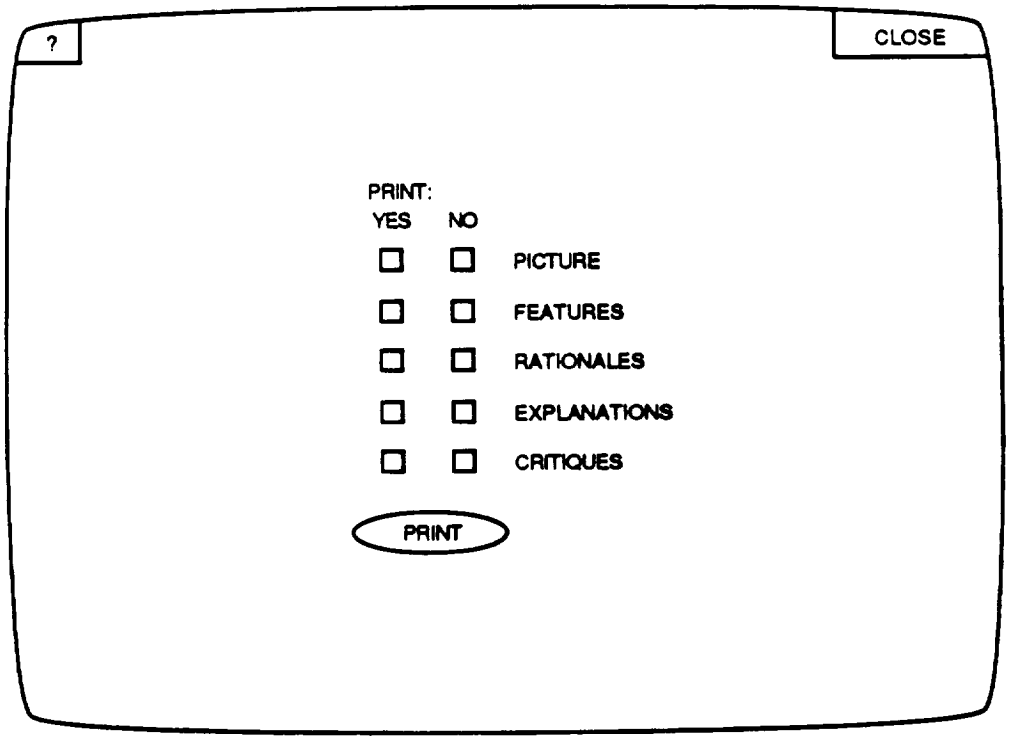

Figure 3-7: Specification Print Options

- *Explanations:* If the *explanations* button is set to yes, the explanations associated with the specification and its features are printed.

- *Critiques:* If the *critiques* button is set to yes, the critiques associated with the specification and its features are printed.

Specification Print Options (cont.)

Clipboard. The Clipboard allows the user to tranfer information between specifications and between items within specifications. There is a single Clipboard per user session, and it is activated from the Home Window using the *clipboard* button. At the end of a session, the user may save the Clipboard and then recall it in a future session.

Items may be placed on the Clipboard in two ways. First, the user may position the cursor on the Clipboard and begin typing. The text will be entered at the current cursor position. This feature allows the Clipboard to be used as a note pad. Secondly, the user may copy items from another window to the Clipboard. This is done by selecting the item using the mouse (in the other window) and then invoking the *yank* button on the Clipboard.

Textual items on the Clipboard may be edited as desired. To add text, position the cursor and begin typing. To delete text, position the cursor and use the *BACKSPACE* key to delete characters to the *left* of the cursor. All types of items (e.g., textual and graphical) may be deleted using the Clipboard's *delete* and *clear* buttons. The *delete* button will delete any currently selected items from the Clipboard, and the *clear* button will delete all items from the Clipboard.

Items may be transferred from the Clipboard to other windows as follows: first select the item on the Clipboard using the mouse; then click the left mouse button at the position in the other window where the item should be inserted; then invoke the *stuff* button on the Clipboard.

Figure 3-8 presents an example Clipboard window. The options available on the Clipboard are described in detail in the table below.

- *Clipboard View:* The clipboard view is a scrollable window that displays the items currently on the Clipboard. Items are added to the view using the *yank* option or by typing text at a selected location. Items are deleted using the *delete* and *clear* buttons, or by backspacing over them.

- *Stuff:* The *stuff* button allows the user to copy items from the Clipboard to another window. The item is first selected in the *clipboard view*. The user then selects the place for insertion in the other window, and lastly invokes the *stuff* button. Several stuffs may be performed in succession without re-selecting the item to be stuffed.

Using KAPTUR's Clipboard

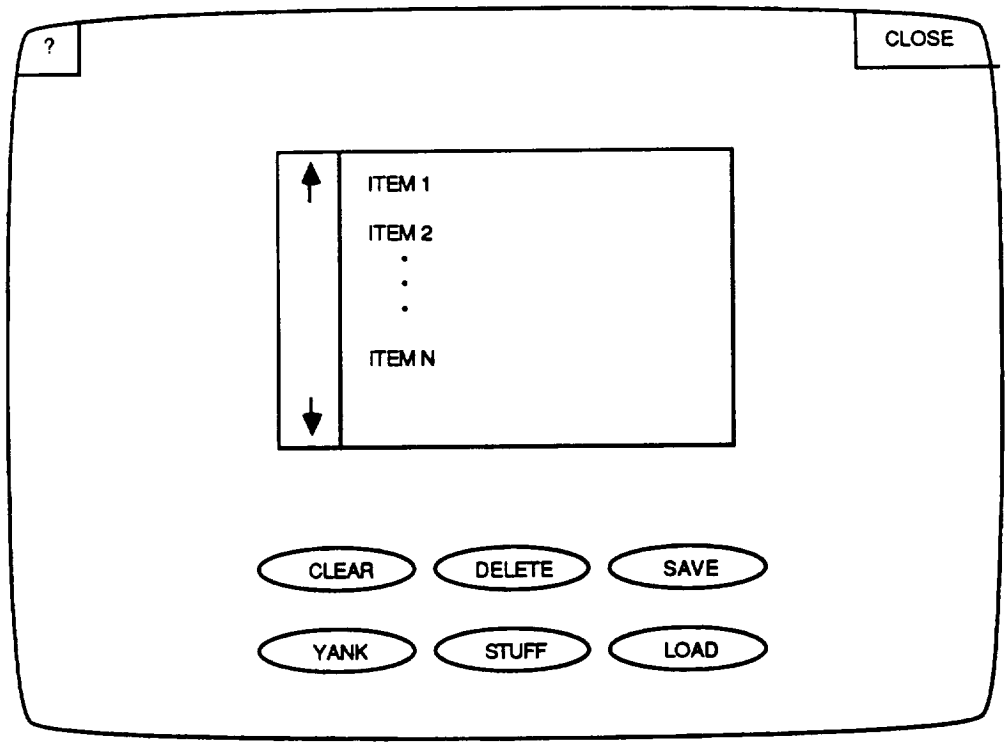

Figure 3-8: Using KAPTUR's Clipboard

- *Yank:* The *yank* button allows the user to copy items from other windows to the Clipboard. The user first selects the item in a window (including possibly the *clipboard view* itself), then selects a place in the *clipboard view* for insertion, and lastly invokes the *yank* button. The same item may be yanked several times without re-selecting it each time. If a position in the Clipboard is not selected, the item will be placed at the bottom of the Clipboard. Each time an item is added to the Clipboard it becomes the currently selected item on the Clipboard. This allows the user to perform a yank followed immediately by a stuff without having to locate the item and then select it again.

- *Clear:* The *clear* button deletes all items in the *clipboard view*.

- *Delete:* The *delete* button deletes the currently selected items from the *clipboard view*.

- *Save:* The *save* button saves the current *clipboard view* to a file. Before the save is performed, a dialog box is invoked for the user to specify the file where the clipboard is to be saved.

- *Load:* The *load* button first invokes a dialog window which prompts the user for the clipboard file to be loaded. It then loads the file into the *clipboard view*. If the current view has not been saved, the user is prompted to perform the save before the load is performed.

Using KAPTUR's Clipboard (cont.)

Distinctive Features Window. A Distinctive Features Window is used to display the distinctive features associated with a specification view. It may be invoked from either a View or an Edit Window. In either case, the features may not be modified since they are determined automatically (see the *features* option under the Edit Window). Only one feature is displayed at a time, but the user may scroll through a set of features using the window's *next* and *previous* buttons.

There are several types of additional information that may be associated with a distinctive feature. For example: other specifications with the same feature, specifications illustrating alternatives to the feature, rationales for the feature, explanations of the feature, and critques of the feature. Each of these may be displayed by invoking the appropriate option from the Features Window (see the table below). If the Features Window has been invoked from an Edit Window, these items may also be modified.

Figure 3-9 presents an example of a Distinctive Features Window.

- *Feature field:* The feature field displays a single feature. It may not be modified. In the present prototype, this will be a textual representation of the feature; in the future it may be changed a graphical representation when that is most appropriate.

Viewing a Specification's Distinctive Features

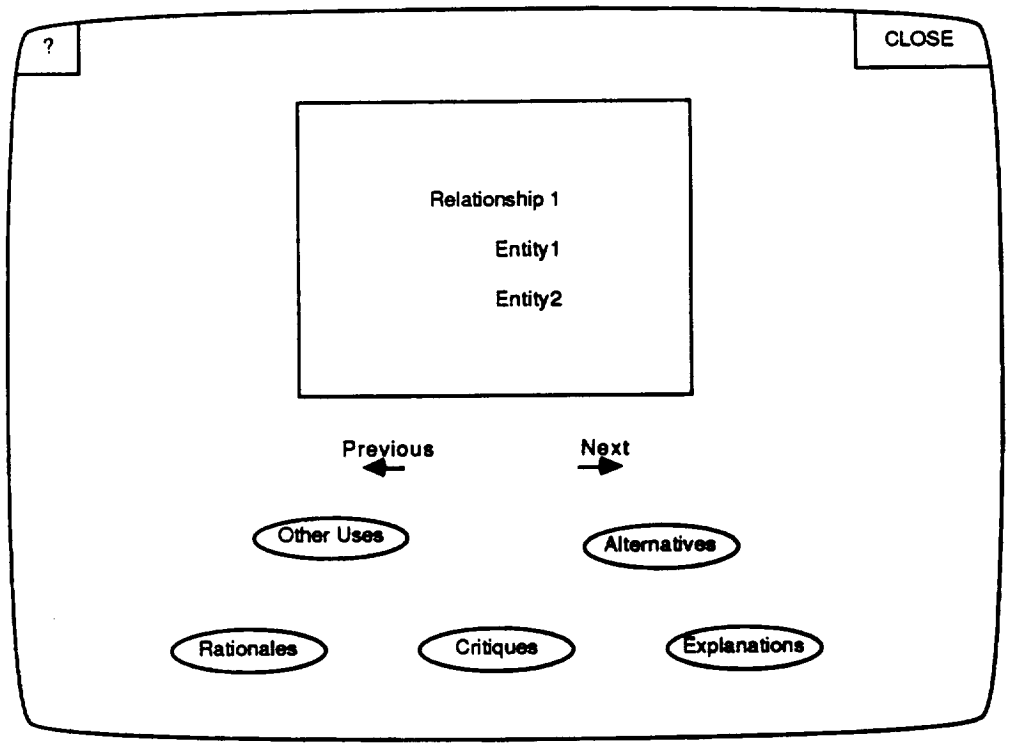

Figure 3-9: Viewing a Specification's Distinctive Features

- *Next and Previous:* The *next* button replaces the current feature in the feature field with the next feature associated with the current view. The *previous* button replaces the current feature with the previous feature. Features are not arranged in any particular order—they are presented in the order in which they were identified.

- *Other Uses:* The *other uses* button invokes a Text Window that displays a list of other specifications with the same feature. From this window, the user may select one (or more) of these specifications to view. If the user invokes the view option, a view window will be opened with the appropriate specification. If the feature can be localized within the specification, the view will be set to display that location.

- *Alternatives:* The *alternatives* button invokes a Text Window that displays a list of specifications that illustrate alternatives to the current feature. From this window, the user may select one (or more) of these specifications to view. If the user invokes the view option, a view window will be opened with the appropriate specification. If the alternative can be localized within the specification, the current view will be set to display that location.

- *Rationales:* The *rationales* button invokes a Text Window that displays the rationales associated with the current feature. One rationale will be displayed at a time. The user may scroll through the rationales using the *next* and *previous* buttons. If the Features Window was called from an Edit Window, the user may edit the rationales.

- *Critiques:* The *critiques* button invokes a Text Window that displays the critiques associated with the current feature. One critique will be displayed at a time. The user may scroll through the critiques using the *next* and *previous* buttons on the Text Window. If the Features Window was called from an Edit Window, the user may edit the critiques.

- *Explanations:* The *explanations* button invokes a Text Window that displays the explanations associated with the current feature. One explanation will be displayed at a time. The user may scroll through the explanations using the *next* and *previous* buttons. If the Features Window was called from an Edit Window, the user may edit the explanations.

Viewing a Specification's Distinctive Features (cont.)

Locating alternatives to a distinctive feature is not as simple as finding a specification that does not illustrate the feature. KAPTUR must help the user understand how the located specification represents an alternative. This is a context sensitive process, depending on the goals of the current user session.

Text Windows. Text Windows are a general class of window that are used for displaying textual items and for navigating the hypertext links between the items in the specification knowledge base. A given text window can display only one class of text at a time. For example. a text window might display an explanation of some item in a specification. However, a text window can invoke other text windows that display other types of information. The user might for example move from an explanation of one item

to other items that use the same explanation. Moving between item types is accomplished through the *jump* option (see below).

The table below describes the options that will be available from text windows in general. In addition to these, a given text window may have several additional options, or it may omit some of the general options. These alternatives are determined by the calling window. For example, the Other Uses text window associated with the Features Window will have a scrollable text field for displaying a list of specification names. It will not have the *next* and *previous* buttons. Figure 3-10 presents a generic Text Window.

- *Text View Field:* The *text view* field is used to display the text associated with the window. It may display a single item within the text, or a complete list of the items. A given text window's view field may or may not be modifiable. These attributes are determined by the calling window.

- *Next and Previous:* The *next* and *previous* buttons (if present) allow the user to scroll through the items associated with the window.

- *Jump:* The *jump* button allows the user to follow the links from the items in a Text Window to other items in the KAPTUR knowledge base. Any type of item in the knowldge base, including specifications, views, functions, rationales, critiques, and explanations, may be accessed by means of a jump. To initiate a jump, the user first selects an item in the *text view*, and then selects the *jump* button. A dialog window then asks the user what type of item to jump to. In response, the user may specify one or more types of items in the system—possibly all types. The KAPTUR knowledge base is then searched for an item that is linked to the selected item and whose type matches one of the specified types. If the search is successful, a window is opened with the located item in it. The type of window opened will depend on the type of item found. If no items are found, a failure message is returned.

Text Window Options

Context Window. The Context Window displays the path by which a user arrived at a given window, and allows the user to return to any window in the path. It is invoked through the *Where* button, which is available on all windows. Figure 3-11 contains an example Context Window; its options are described in the following table.

- *Path View:* The *path view* is a scrollable text window that describes the path by which the user arrived at the current window. The description is a list of window types and item names. The item name defines the item being viewed in the corresponding window.

- *Last:* The *last* button closes the current window (before the Context Window), and reopens the last window listed in the path view. If the last entry is the Home Window, the current window is simply closed.

Context Tools Help the User Avoid Getting Lost

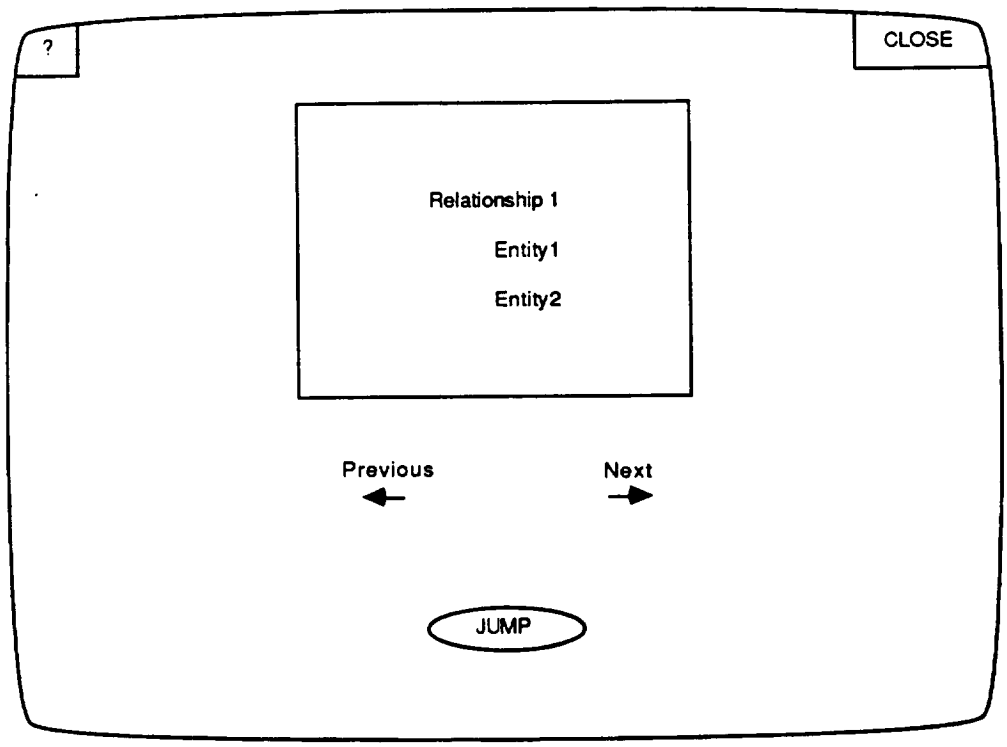

Figure 3-10: Text Window Options

- *Previous:* The previous button allows the user to return to any window listed in the path view. Before invoking the option, the user must select an item in the path view. Once the option is invoked, all windows between the current window and the selected window are closed, and the selected window is reopened. If open files are detected, the user will be prompted to save them before the window is closed.

- *Home:* The home button deletes the current path and returns the user to the Home Window. Any windows that have been opened along the path are closed. If open files are detected, the user will be prompted to save them before the window is closed.

- *Continue:* The continue button allows the user to reinvoke the last jump operation to continue a search. When continue is used, KAPTUR searches the current context path to see where the search has been performed before and what results were viewed. It then omits these results as solutions to the continued search. In effect, this feature allows the user to see the first item with a given set of properties, then the second, and so on.

Context Tools Help the User Avoid Getting Lost (cont.)

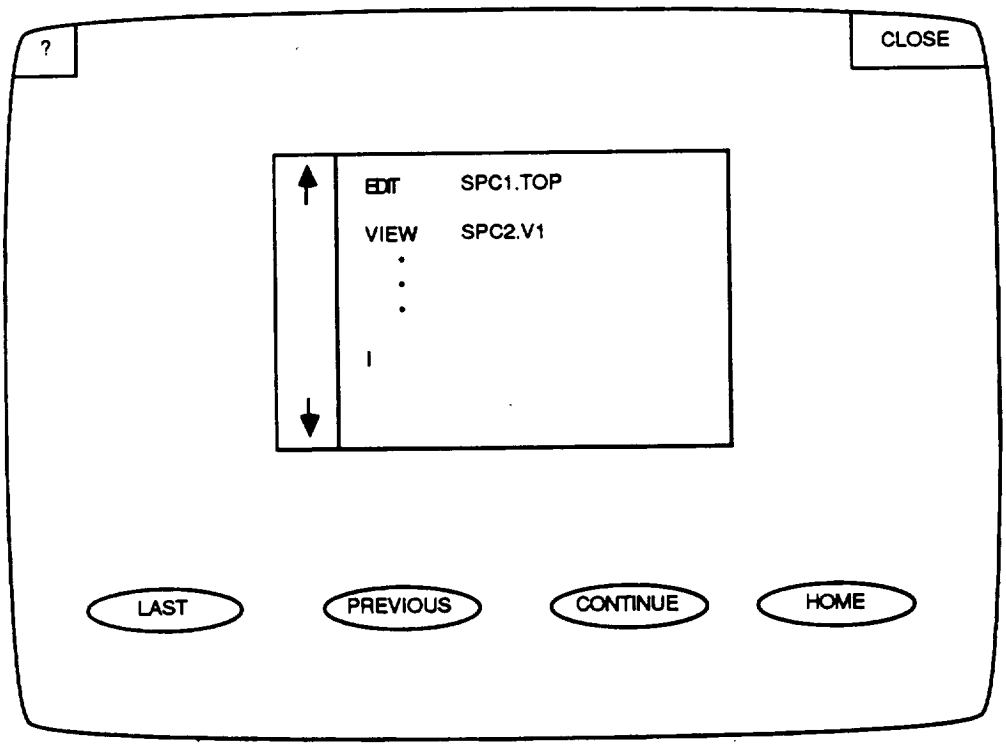

Figure 3-11: Context Tools Help the User Avoid Getting Lost